Heikki Nykyri

Implementing Continuous Availability in Java EE Environment

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

28 February 2018

Helsinki Metropolia University of Applied Sciences

Author	Heikki Nykyri
Title	Implementing Continuous Availability in Java EE Environment
Number of Pages	45 pages + 3 appendices
Date	28 February 2018
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Antti Koivumäki, Principal Lecturer Timo Pessi, Director of Product Development

Continuous availability is a highly desired functionality in the field of information technology, especially in the computer systems. Continuous availability means that there should be no service breaks which users can see. For example, users should be able to use the system even when there are updates going on.

The case company Elisa Appelsiini Oy has the system running on the Java EE environment. The system´s user base has grown steadily and currently the system is being used around the clock. The current environment does not support software updates without a service break. That is why it was crucial to develop and improve the current environment towards continuous availability.

The main problem was to make the environment support updates that users cannot see. Another big issue was to change the deployment process on how the application is delivered to the production. To overcome these issues, it was investigated how to update the application without a service break. There was also a need to make changes to the deployment process to support application updates. Fortunately, the Java EE platform supported updating applications without the service breaks if the application met certain requirements. The deployment process change was necessary to support the application updates. There were limits to how the application could be updated and to tackle those limits the deployment process to support small updates was made. The automated tests and an environment for the tests to increase the speed of production updates were added.

There were multiple applications running on the Java EE environment and this thesis was scoped to concern just one application. The rest of the applications can be transformed later. This design guideline was created simultaneously with the implementation. The design guideline lists all necessary requirements the application must meet to support updates.

The result was successful and the application can be updated without a service break. The key finding was how difficult it is to change an old deployment process to a new one. Mainly it was people's mind sets that had to be changed which turned out to be quite hard. Eventually with good reasoning the new deployment process was taken in use.

Keywords	Continuous Availability, Java EE, WebLogic, Docker, Automated testing

Contents

Abbreviations and acronyms

Arquillian	Is an innovative and highly extensible testing platform for the JVM that enables developers to easily create automated integration, functional and acceptance tests for Java middleware.
CA	Continuous availability, is a method or an approach which makes the system more fault tolerant so that users do not see interruptions when using the system.
CD	Continuous delivery, is a method which consists several practices to ensure that application can be deployed to production safely.
Continuous deployment	Is a method where every approved application version is deployed to production. Usually continuous deployment is considered next step from continuous delivery.
CI	Continuous integration, is a practice where developer should continuously (several times a day) share their code between other developers.
CIFS	Common Internet File System, is a protocol for sharing file access across network, usually corporate's intranet.
Continuous integration server	Is a server which usually runs tests and other metrics automatically when developer commits code to version control system.
Docker	Is a platform for Docker containers.
Docker container	Is a way to isolate packaged software from outside operating system. Docker containers are used along Docker platform.
EAR (Java)	Enterprise Application Archive, is a file format for Java applications. Usually it contains multiple modules (JAR, WAR, etc) and deployment descriptor in XML format.
EJB	Enterprise JavaBean, is a server side component architecture for Java EE.

HA	High Availability, is a system characteristics which usually means that system should be almost always running and available.
JMeter	Is an open source application for load testing functional behaviour and measure performance.
JMS	Java Messaging Service, is a API (application programming interface) that allows applications to create, send, receive and read messages.
JSP	Java Server Pages, is a technology used to create dynamic web pages.
LAN	Local Area Network, is a network which connects computers together in a relatively small area.
Maven	Is a software project management and comprehension tool. With Maven it is possible to build, generate reports and documentation from a central piece of information.
Maven pom.xml	Project Object Model, is a XML file where is defined how Maven should build the project.
NAS	Network Attached Storage, is a data storage server in network. It provides data store services to network users.
NFS	Network File System, is a distributed file system protocol. It allows users to use network storage almost like a local storage.
SAN	Storage Area Network, is a network which provides access to block level data storage. Compared to NAS, main difference is that SAN uses block level access which means that file system concerns are left to client side.
WAR	Web Application Archive, is a file format for storing different types of files as one single archive. This archive is usually called web application.
WLST	WebLogic Scripting Tool, is a command line scripting tool for operating WebLogic Server instances and domains.

1 Introduction

This thesis is about transforming one Java application running on an existing Java EE environment to be continuously available. At the same time implementation guidelines for other applications running on the same environment were created. Implementing continuous availability to the other applications in that environment is out of the scope of the present study. Scoping was done because there were over ten other applications running on that environment and not all applications are under the case company's control, so it might be difficult or even impossible to make changes to those.

1.1 Background

Continuous availability (CA) is a highly desired feature in the field of information technology. What continuous availability means is that there cannot be almost any interruptions on the service. Usually when talking continuous availability, "Five 9s" are mentioned. This means that the service should be available 99,999% of time. In a year this means a little over 5 minutes. (Alcott, 2010) Basically, the service should be always available. When talking about continuous availability, the term high availability (HA) is usually involved in some way. The high availability term could be described as follows: "High availability is the characteristic of a system to protect against or recover from minor outages in a short time frame with largely automated means." (Schmidt, 2006, p. 22). Mainly high availability differs from continuous availability by availability time. In HA availability is usually "Four 9s" respectively 99,99%. In a year, this means a little under 53 minutes. In other words, continuous availability system cannot practically have any service break when a high availability system can have one or two in a year depending on the length of break. High availability can be thought as a subset of continuous availability.

To the service providers, continuous availability is more costly than high availability. Building systems that can handle unexpected failures, so that the user does not see any interruptions, is more expensive than building an ordinary system where failures usually means interruptions to the users. That is why it is always about balancing between growing costs and how highly/continuously available the service should be.

In Java EE environment availability usually means the application availability. The application should be running and available without major service breaks. There are also assumptions that the underlying infrastructure supports continuous availability and is built with it in mind. Of course, Java EE applications are running inside the application server so there is a need to think continuous availability of the application server also. Usually this means just clustering and load balancing. To take one step further from clustering and load balancing there can be the backup servers or different update servers which makes it possible to have continuous availability. The different servers offer the possibility to update newer version of the application without interrupting users. Nowadays, systems are usually made to be available all over globe. Systems are used 24/7 and there is not a good time to have such service breaks. That's why it's important to have a continuous available system to serve customers.

1.2 Case Company and Technology Challenge

The case company of this study is Elisa Appelsiini Oy. Elisa Appelsiini is the Elisa IT business unit. It has over 350 employees, over 120 000 cloud service users and 200 IT outsourcing hosting service customers.

The case company has an application running on Oracle WebLogic server (Java EE environment). Its availability times have expanded from the working hours to 24/7. Before it was easy to update the software just outside working hours. Currently they need to update the software at night time and still there are users who suffer from those updates. Therefore, the research objective to this topic is to investigate the continuous availability concept and implement it to our Java EE environment based on the study.

The application is also critical to Elisa Appelsiini's customer so it must be available if not all the time then at least most of the time. That is why there is a clear demand to build the system with as much continuous availability as possible. To build such system, it requires deep knowledge of the software which are running in the environment. It also requires studying platforms best practices and maybe think different ways of environment implementation.

1.3 Objective and Outcome

The study aimed to transform existing Java EE environment to be continuously available. And implement the continuous availability to the one application which was running in the environment.

The outcome of the project was to have one application supporting continuous availability, running on the existing Java EE environment and the design guidelines how to implement continuous availability to the Java application running on Oracle WebLogic server.

1.4 Research Method

To develop the solution there was a need to study the best practices of the running Java EE environment and gather general knowledge of the both high availability and the continuous availability. Second, the current environment needed to be explored. And based on that investigation, changes to environment how to achieve these availability requirements, was decided. Third, the plans and the guidelines for the continuous availability and the implementation need to be created and written. Finally, based on the plan, the solution had to be implemented in to the running environment. To verify that the solution was working, there was a need to build a script which creates HTTP sessions to the application. The script should write a log if creation fails. Then the application was updated and after that the session creation failures were measured. The script was run before and after the environment change. After that, verification was made to see were there any improvements to previous solution. HTTP session creation should drop to zero during the updates after the implementation has been applied.

The process of the project is described in the following steps:

1. Exploring existing literature. The literature consists general research papers, previous studies and books about the high availability and the continuous availability in general. There were also essential papers about best practices in how to do proper implementation to the existing environment.
2. Investigating the current state of the target environment.
3. Based on the results from analysis in step 2, the developing implementation guidelines on how to do a proper implementation.

4. Implementing the solution to the development and the test environments according to the guidelines.
5. A review / testing phase evaluation of the implementation.
    a. If everything is working as expected continue to the production implementation, step 6.
    b. If not, find out what is wrong, fix it and adjust the guidelines (go back to step 3). This fix - test iteration (from step 3 to 5) is done as many times as needed.
6. After the tests have passed the final step is to apply the implementation to the production environment and finally verify that the production is working properly.

This study is written in eight sections. Section 1 is this introduction. In Section 2 the used resource methods and design are explained. Section 3 contains the existing knowledge of the high availability and the continuous availability, it also describes the current environment. Section 4 concentrates on the existing knowledge of the continuous delivery and in Section 5 the background of the Oracle WebLogic Server is explained. Section 6 describes the actual implementation. Section 7 contains the results and analysis. Final section, Section 8, is the conclusion where the thesis is summed up.

2 Materials and Methods

The case company has a clustered Java EE environment running on dedicated hardware. No other virtual servers or any other virtualization solutions were used. An Oracle WebLogic application server runs on that hardware. In that application server, there are multiple server instances for different Java applications. Some of the Java applications are several years old and some are brand new. There is a wide range of technology and techniques used on the applications. Of course, the oldest were modified to run on the current application server but the old Java EE version is always there and this makes it even harder to maintain such software.

The environment itself is mission critical to the customer and that is why it is very important to have all services running as smoothly as possible. Of course, there will be a need to make changes to the running software, add new functionality, fix bugs or any other changes. These software changes usually cause (it depends on what kind of update it is going to be) a service break to its users. The customer always wants as few service breaks as possible and that is why there are maintenance windows when software (or other system) updates can be done. Earlier the maintenance windows were after working hours, usually after 5pm local time. The system gains more users all the time and the system usage has grown from working hours to 24/7 usage. That makes maintaining the system more difficult because there is not a good time in a day to do updates. Currently updates are made at night time and even then, there are users who suffer from the service break. The update windows are kept as short as possible but if multiple software updates are made the break is going to last.

That is why the current environment and its software needs to be updated and modified to allow software updates to be performed without service breaks. This kind of software is usually called continuous available software. It means that even upon an update process the user is able to use the software and in the best case scenario the user does not even notice that the underlying software is being updated.

2.1 Context and Outcome

This thesis was done alongside a company project and the purpose was to update almost all previously mentioned Java applications running on that environment to support continuous availability. The Java applications represents multiple Java versions and frameworks which made it quite hard to do proper implementation. That is why the thesis only includes one application which is maybe the most used application in the environment. Other applications were updated after the present study and as said those are out of scope.

The outcome of the study includes one application running on the previously mentioned Java EE environment and it supports continuous availability. Another outcome is the guidelines which describe what has to be done to make such software to support continuous availability. With the guidelines the company were able to start implementation for other Java applications to be continuous available.

2.2 Research Design

This thesis research design follows the design science research approach. This basically means that something is going to be built and after building there is an evaluation phase for evaluating the built result. If the evaluation is not successful, the building phase is performed again and after that there is evaluation again. This will continue as long as the evaluation will pass so this is an iterative method.

A proof of concept implementation was built to the development environment, and the evaluation of the solution was done. The proof of concept implementation was working as expected. That meant that the first acceptance criteria were fulfilled, and implementation was applied to the testing environment. Evaluation on the testing environment was also successful which meant second that production installation could be made. Along with the build/evaluation phases, guidelines to how implementations should be done, were written

For the evaluation, the JMeter script was written to measure the continuous availability. The JMeter is designed to load test functional behavior and measure performance (Apache Software Foundation, 2015). JMeter can be used without load testing to just do necessary application calls. This is how it were used here. In this case the JMeter script called the application and it calculated user HTTP session creations and measures

failure count when updating the software. After successful implementation failure count of sessions were zero. This meant that the application can be updated without interfering users.

3 High Availability and Continuous Availability

This chapter explains the general concepts of high availability and continuous availability. At the end of this chapter there is also explained the current environment status.

3.1 High Availability

High availability in general is such a broad subject that there are many books written about it. That is why here it is explained only briefly related to the present study and to the current environment.

As Klaus Schmidt describes. a high available system can have service breaks for various reasons but the point is that a system is expected to recover itself in a short time frame. (Schmidt, 2006, p. 22) That is why it is not catastrophic to interrupt user as long as the user can continue using the system for a while and continue what he or she was doing.

In general, the main point in high availability is that there should not exist single point of failure which can cause a service break. This means that almost every component should have a duplicated version or a backup equivalent. For example, thinking of an application server, there could be a server cluster where the application server is installed. That makes it possible for the servers to fail and the service is still functional. Maybe with not the same capacity but the users can still use the system. Same goes to every component which are needed on the service. Of course, if there is a component which is extremely expensive there can be a decision that this single point of failure component is acceptable. In this case, there should be plans on what can be done to prevent failure and what is done when the failure happens. In high availability, some service breaks are accepted as described earlier.

High availability in servers is usually implemented using clustering technology. The cluster may contain two or more servers which are communicating together with some protocol. It is also possible that the cluster nodes do not know existence of the other nodes. The clusters can be active/active or active/passive types. Active/active cluster type means that the all servers which are in the cluster are active (running) all the time. Requests are forwarded to nodes using some algorithm. Usually algorithms spread the load equally between nodes.

Active/passive type of cluster means that there are one or more cluster nodes which are in a passive state. Inactive servers do not handle requests. They are there only to be started up if one or more active servers fails. In Figure 1 is shown both active/active and active/passive type clusters.

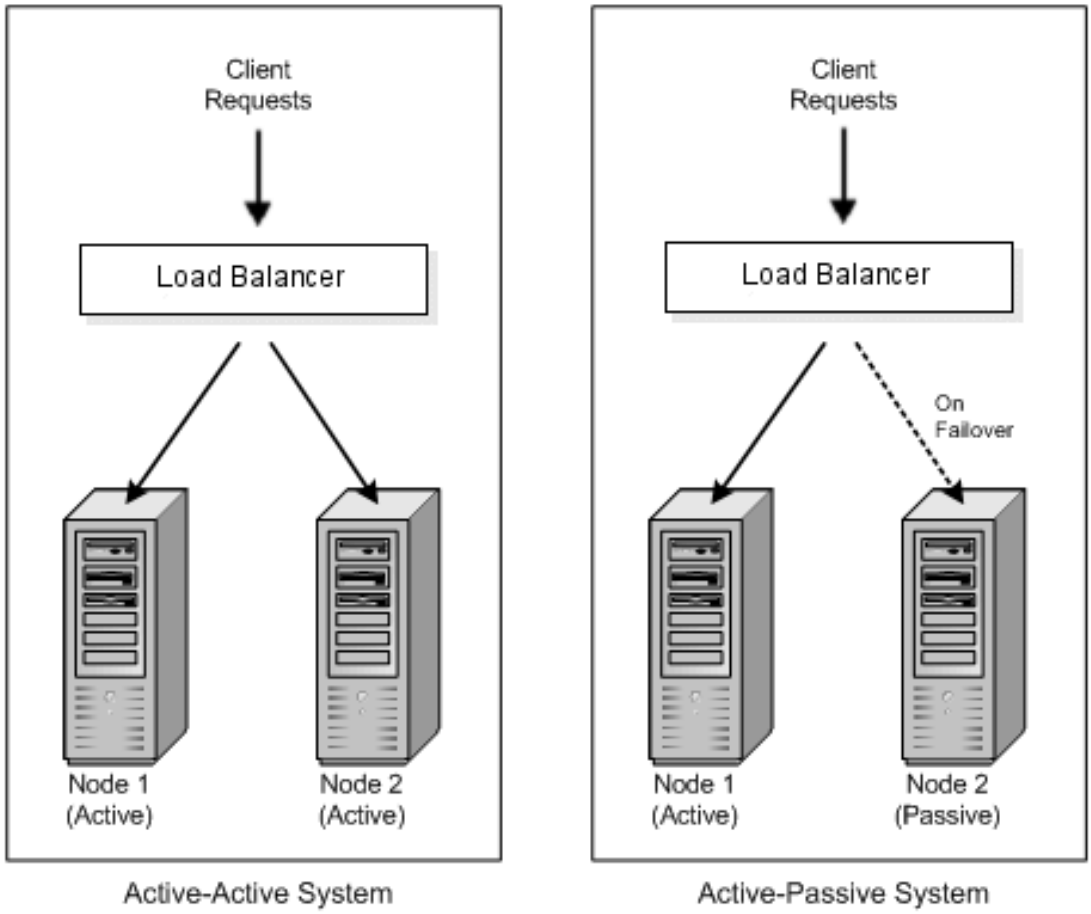

Figure 1: Active-active and Active-passive High Availability solutions (Oracle, 2005)

Another important hardware configuration in the high availability environment is the shared storage. It makes it possible to share stored information between servers. For example, information could be server configuration or it could contain dynamic state of the services. Example of a dynamic state could be file based persistence stores for JMS. Shared storage is usually configured to use NAS or SAN networks. These two networks provide its storage devices via network such as LAN. One difference between these two technologies are how they handle actual files. In a NAS the storage devices are directly connected to a file server that makes the storage available at a file-level to the other computer (Wikimedia Foundation, Inc., 2017). In a SAN, the storage is made available at a lower "block-level", leaving file system concerns to the "client" side (Wikimedia

Foundation, Inc., 2017). This means that the SAN is visible to client operation system as a disk when the NAS is visible as a file server. So, the SAN can be formatted and mounted by client operation system whereas the NAS is mapped as network drive.

In the WebLogic server, there are typically three different artifacts which are placed to the shared storage. In Oracle Fusion Middleware High Availability guide (Oracle, 2014, pp. 3-1) those are described as follows:

- **Product binaries**: All files and directories related to product executables, JAR files, and scripts that install during product installation.
- **Domain directory**: The directory containing the WebLogic Server domains and their configuration.
- **File-based persistent stores**: File-based persistence stores for JMS persistence and JTA transaction logs.

The Oracle Fusion Middleware product binaries are stored into an Oracle home. These binaries are read-only and typically changed only when patching or updating server software. Usually product binaries should be a same version in the all server nodes. This makes the product binaries as excellent candidate to store shared storage. This also saves disk space when binaries are located in one place. Downside to this approach is that the server software cannot be updated one server at time (and avoid service break). All servers which use these shared binaries must be turned off at same time.

In the domain directory approach, the Administrator server configuration files or the Managed server configuration files are saved to the shared storage. Saving Administrator server configuration files makes it possible to recover from server crash just to mount the shared storage to new the server and start the Administrator server from there. Oracle recommends keeping the Managed Server configuration files in local, or, host private, storage (Oracle, 2014, pp. 3-4). Albeit it is possible to store such files in the shared storage. In that case performance issues should be taken into account. It could be a major performance hit if all managed servers are reading the configuration files from there.

In a file-based persistence JMS persistence and JTA transaction log directories must be configured to use the shared storage. It is mandatory that all the servers can access the stored information. When using the file-based storage and a NFS there are a couple of things that need to be considered. First, which NFS version should be used and

secondly, should file locking be disabled on that share. More information can be found from these considerations at Oracle Fusion Middleware high availability guide (Oracle, 2014, pp. 5-3 - 5-6).

There are also lots of other techniques and technologies how the high availability can be achieved. It really depends on environment where it is planned to implement. Is the environment virtual or is dedicated hardware used? What is the operating system etc. Previously mentioned techniques are used in the environment used in the present study.

3.2 Continuous Availability

Continuous availability is a broad subject and is covered here only briefly. Compared to high availability, continuous availability has the same purpose, to have as few service breaks as possible. The difference is that there are far fewer breaks allowed in continuous availability. Practically there cannot be any service breaks at all.

Continuous availability also encompasses same kind of requirements as high availability but the requirements are stricter. This means that there cannot be any single point of failure. Every single component must be replicated or there must be another way to perform failed task. Because of these tighter requirements implementing continuous availability is more expensive than implementing high availability.

For example, usually when clustering the environment, the high availability environment could have active/active cluster node configuration. In continuous availability, active/passive cluster type should be considered as primary solution because it has better failover options. It provides same kind of user experience when thinking about the server loads. When one or more servers fails the inactive server is started to serve users. Overall the server load is constant because the server amount is always the same.

3.3 Current Environment Status

The current environment is clustered with two application servers and two database servers. The cluster is active/active type of cluster which means that the both cluster nodes are in use. There are no backup or reserve servers. All the servers have dedicated hardware so no cloud or virtualization technology has been used. The servers are located in two different locations to avoid problems in one physical location. The

database is Oracle Database. The database servers and software are designed with high availability in mind. This is achieved by using Oracle Real Application Cluster (RAC). The application server software is Oracle WebLogic which has been configured as cluster. This means that the application server is also designed with high availability in mind.

The database connections have been configured in the WebLogic server. Every application configuration contains a connection pool for the application connections to the database. The configuration is also configured as load balanced which means that the requests are forwarded to every configured database nodes. Other option could be failover where the requests are forwarded to a single node and in failure case requests are forwarded to another node. Because clusters are working as active/active there is no need to use failover mode in these connections.

In front of the cluster there is a load balancer. The load balancer will forward requests to each server using its own proprietary algorithm. If one node fails to respond to the request the load balancer forwards that same request to another node. Failure in one node is not catastrophic but it should be fixed as soon as possible. Using only one server causes slowness.

Each application servers have HTTP server forwarding requests to correct application. HTTP server is an Oracle HTTP Server (OHS) which is based on Apache HTTP Server. Oracle has added some security features to it but mostly it is Apache HTTP Server. HTTP server is also serving some static files. Files are located in a shared storage so the both application server nodes can see the same files.

The shared storage in the current environment is NFS mounted SAN storage. Both servers can access to it and it replicates files between nodes almost instantly. The shared storage is used for the WebLogic administration console domain files and for the JMS persistent store. Some other application specific configuration files, which can be shared between server nodes, are also stored there. The application log files are stored there too. The shared storage also makes it a bit easier administer the environment. Files can be shared easily between nodes and all the application log files can be examined from all the servers.

The current environment is mostly made with high availability in mind but there are still many things that can be improved. The main issue in the present study is how to update

the software without a service break. Another thing is how to patch the application servers without a break. Currently patching without the service break is not supported on the WebLogic server version that is used. That would require updating the WebLogic to a newer version. Version upgrade is not included here. There are also lots of minor configuration and software development issues on the software running on the WebLogic but those are also out of the scope of this thesis.

Thinking about continuous availability, the current environment is quite close to it. As the previous chapters describe, the current environment is mostly high available and that is a key element for achieving the continuous availability. What needs to be done is to make software updates seamless so that users cannot see any interruptions. Luckily Oracle WebLogic server supports that out of the box. This feature is called Production Redeployment. With that feature and some build process change it is possible to achieve continuous availability in the current environment. In Figure 2 is shown an overview of the current environment.

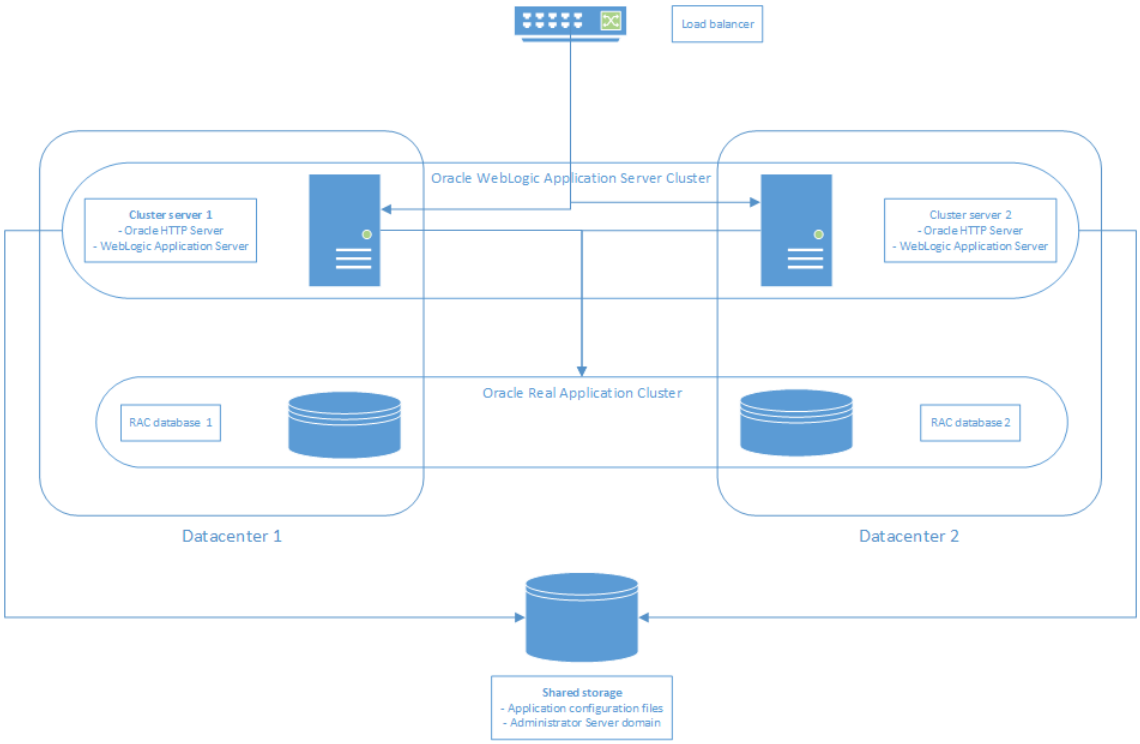

Figure 2. Environment overview

On the top there is the load balancer for handling incoming requests. It distributes requests to two servers, cluster server 1 and cluster server 2. The servers are located in two different data centers. The data centers are located geographically in different

locations. The both cluster servers contains the HTTP server and the Oracle WebLogic application server. The HTTP server purpose is to forward requests to the Oracle WebLogic servers (to both cluster servers). The forwarding is done by Oracle's proprietary HTTP server plugin. The Oracle WebLogic servers are configured working as the cluster. Each data center also has the database servers, RAC database 1 and RAC database 2. At the bottom there is the shared storage which is available from the both cluster servers.

4 Continuous Delivery

One big problem in the software industry is how to deliver software as quickly as possible. To solve this problem there is a process or approach called continuous delivery. It is a method for releasing software quickly, constantly and reliably. The continuous delivery or CD is a delivery process which can be followed to release new versions of software and make the release process easier. The continuous delivery pipeline usually contains phases from making the code to releasing it.

In Figure 3 there is an example of a process continuous delivery can follow.

Figure 3. Example of continuous delivery process

The continuous delivery process starts from coding. The team will make a desired function to the application and push it to the version control system. After pushing the code, the continuous integration process will start it tasks. Usually the continuous integration process contains tests (unit, integration, acceptance, capacity, etc) running and deployments to the corresponding environments. Tests in the continuous integration server are usually automated. Finally, there might be manual verification for implemented functions before the actual release. Manual verification could be testing user interface by doing exploratory testing or just showcasing it to someone. In any phase, this delivery process could stop if required. Usually it also generates alerts for someone to look after it. A reason for the interruptions could be that a test has been failed or configuration was incorrect. (Humble & Farley, 2011, pp. 111,112)

The continuous delivery process is highly automated so that if everything goes as expected there is not any human interaction. Of course, in the manual testing phase (if there is one) human interaction is required. The final phase, release phase, is not automated at all. Usually there is a person who decides if the release is going to production or not. That is why every release is not necessarily installed to the production.

When the production deployment is also automated, then it is usually called continuous deployment.

4.1 Current Status

The current build process slightly conforms the continuous delivery process. The main difference is in the continuous integration (CI) and the release parts. The continuous integration is implemented in the pipeline but it does not break the IT. This means that if the user interface (UI) tests fail the release build can still be made. Programmed UI tests are not that reliable so there is lack of trust to them. Usually when a failure occurs tests are run again and a decision is made on whether the reason for the failure is something serious or not. Usually after running the tests a couple of times, they eventually pass. Also, there are no automatic production releases. The release cycle is one release a month. So, the release process is quite traditional. Releasing only one release a month is also practical. Usually there are lots of changes and testing them takes a long time (including manual testing). That is, it would be better to release smaller pieces more often.

4.2 Improvements to Current Process

There are a couple of places where the current process can be improved. First, making tests more reliable. It is crucial to have reliable tests which guarantees that the new release does not break anything. In general, regression bugs can be avoided with automated tests and it also helps overall maintainability of application. The second thing to do is to change the process to allow smaller releases. The third thing is to improve the release cycle by changing it to allow to do releases more often.

5 Oracle WebLogic Server

In this chapter there is a brief introduction to technologies that are closely related to Oracle WebLogic Server and an introduction to the application server itself.

5.1 Java Enterprise Edition

“Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals.” (Oracle, 2015) Basically, Java EE is one part of the Java Platform where different smaller components or technologies are collected. Typically, enterprise edition is used on commercial and large applications. Of course, it can be used in any size of applications. But the enterprise edition contains a lot of technologies so there is a chance that it contains too much for simple use cases. Included technologies are for example: Java Servlet, Java Server Faces and Java API for RESTful Web Services.

In Figure 4 is shown the Java EE 6 architecture where different components are structured as an integrated stack working together.

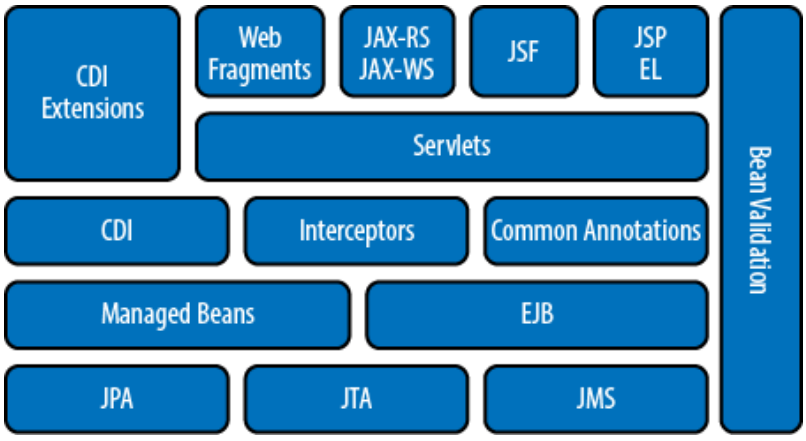

Figure 4: Java EE 6 architecture (Gupta, 2012)

At the time of writing this thesis Oracle and Java community moved Java EE from Oracle to Eclipse Foundation. Java EE got also new name EE4J, Eclipse Enterprise for Java. “EE4J is based on the Java Platform, Enterprise Edition (Java EE) standards, and uses Java EE 8 as the baseline for creating new standards.” (The Eclipse Foundation, 2017)

Because EE4J is based on Java EE 8 and current environment is using Java EE 6, Java EE term is still used here.

5.2 Java Application Server

The Java Enterprise Edition application typically runs on the Java application server. The Java application server is a framework for applications. It provides necessary components to the applications such as web server for web applications, EJB container for EJBs and depending on the application server it could provide also fail-over and load balancing features. Specially for the Java application servers there are technologies such as servlets, Java Server Pages (JSP) and Java Message Service (JMS).

There are many different Java application servers for different purposes. One popular open source application server is WildFly (formerly JBoss AS) which provides all previously mentioned technologies (and many more). Another popular open source Java application server is GlassFish. When considering commercial Java application servers there are three major options, Oracle WebLogic server, IBM WebSphere and JBoss Enterprise Application Platform (EAP) from RedHat. JBoss EAP is the only open source application server from the previous three servers.

The Oracle WebLogic server is used in this environment and is therefore introduced below.

5.2.1 Oracle WebLogic Server

The Oracle WebLogic server is a Java Application server made by Oracle. It is a commercial and closed source server albeit it can be downloaded from Oracle website for free. It is free for developers but to use it any other way a license must be bought. Oracle offers three different editions from WebLogic server, standard edition, enterprise edition and suite. Standard edition provides base technologies to run Java enterprise application. These are for example support for Java EE 6, Java SE 6 and 7 certifications, Oracle Web Tier and many more. Addition to standard edition features, enterprise edition provides support for high performance clustering and failover capabilities, dynamic cluster and Java SE Advanced, e.g. Java Mission Control and Java Flight Recorder application monitoring and diagnostic. The most comprehensive edition is suite. It includes all features from standard and enterprise editions and in addition to that it

provides Oracle Coherence Enterprise Edition, Active GridLink for RAC and Java SE Suite for minimize application latency. (Oracle, 2014)

In the beginning, WebLogic Java application server was developed by a company with a same name in 1995. It was standards-based Java application server. In 1998 BEA Systems acquired WebLogic, Inc. and WebLogic server becomes BEAs product portfolio. From 1998 to 2007 BEA Systems developed WebLogic until Oracle acquired BEA Systems. After that WebLogic server has been in Oracle's product portfolio. Currently latest version of WebLogic server is 12c (12.2.1) and it provides full Java EE 7 support, Enterprise JavaBeans 3.2 and clustering and high availability support for WebSocket 1.1 applications, just to mention a few.

5.2.2 High Availability in Oracle WebLogic Server

Oracle has own recommendations on how to implement high availability in WebLogic Server. Referring to Oracle's best practices (Oracle, 2014) there are eight different concepts available:

1. Server Load Balancing in a High Availability Environment
2. Application Failover
3. Real Application Clusters
4. Coherence Clusters and High Availability
5. Disaster Recovery
6. Install Time Configuration
7. Application and Server Failover
8. Roadmap for Setting Up a High Availability Topology

In the following chapters these concepts are briefly explained.

5.2.2.1 Server Load Balancing in a High Availability Environment

This concept base idea is that a third-party load balancer should be set up or use Oracle HTTP Server or Oracle Traffic Director in front of the servers. This way it is possible to manage how traffic is forwarded to the servers. In this concept Oracle also provides requirements for load balancer and how to configure it. It also describes how to user Oracle products to provide load balancing.

5.2.2.2 Application Failover

This concept idea is that when there is an application component error (component becomes unavailable) the application should copy failed component state and start a new component which handles the rest of remaining job. A user should not see any interruption. In this concept, there is a list of requirements what should be met to have such functionality.

5.2.2.3 Real Application Cluster

The Oracle Real Application Cluster (RAC) concept is for the databases. This concept describes that clustering the database is the key and also what the Real Application Cluster is. It also describes what benefits there are when using clusters and shows where more information about RAC can be found.

5.2.2.4 Coherence Cluster and High Availability

A Coherence cluster is a collection of Java Virtual Machine (JVM) processes running Coherence. In WebLogic server version 12c, these processes are referred to as WebLogic Managed Coherence Servers. JVMs that join a cluster are called cluster members or cluster nodes. (Oracle, 2014, pp. 2-5) Basically, the Coherence Cluster is Oracle's product for managing WebLogic servers in cluster. It has its own communication protocol (Tangosol Cluster Management Protocol, TCMP) and they have special configuration file (Grid Archive, GAR).

5.2.2.5 Disaster Recovery

This concept is about distributing the system in different geographically locations. It will help recovering for example from natural disasters. Oracle's products support configuration where the servers in different location can act as a backup server.

5.2.2.6 Install Time Configuration

Install Time Configuration concept has two different profiles. The Domain (Topology) Profiles and the Persistence Profiles. The first profile describes instructions to install WebLogic 12c using the expanded profile and it provides links where to seek more additional information. The latter profile describes where to store different component /

service information. This profile also provides links where to seek additional information about configuration.

5.2.2.7 Application and Service Failover

The Application and Service Failover concept describes two topics, Whole Server Migration and Automatic Service Migration. The main difference in these topics are what to migrate to a new server. The first topic means that the whole server is migrated to the new server when current server is not capable to serve requests. The latter topics is just for pinned services which can be migrated to the new server instance.

5.2.2.8 Roadmap for Setting up High Availability Topology

Roadmap concepts has a clear task list what should be performed to set up a high available system. It points five different task and documentation for them. The tasks are:

1. Install Real Application Cluster
2. Install middleware components
3. Install Oracle HTTP Server
4. Configure a load balancer
5. Scale out the topology (machine scale out)

5.3 Application Redeployment in Oracle WebLogic

Application redeployment in Oracle WebLogic server means how to reinstall and start existing application on a server. There are several ways how application redeployment can be performed in Oracle WebLogic server. These methods are:

- In-Place Redeployment of Applications and Modules
- Partial Redeployment of Static Files (In-Place Redeployment)
- Partial Redeployment of Java EE Modules (In-Place Redeployment)
- Production Redeployment

In Table 1 is summarized the four different redeployment strategies.

Table 1. Summary of Redeployment Strategies. (Oracle, 2015, pp. 8-3)

Redeployment Strategy	Summary	Usage
Production Redeployment	Redeploys a newer version of an application alongside an existing version of the application.	• Upgrading Web applications and enterprise applications that demand uninterrupted client access.
In-Place Redeployment of Applications and Modules	Application classloaders are immediately replaced with newer classloaders to load the updated application class files. WebLogic Server does not guarantee uninterrupted client access during redeployment, and existing clients' state information may be lost.	• Replacing applications that have been taken off-line for scheduled maintenance. • Upgrading applications that do not require uninterrupted client access.
Partial Redeployment of Static Files (In-Place Redeployment)	HTML, JSPs, graphics files, or other static files are immediately replaced with updated files.	• Updating individual Web application files that do not affect application clients.
Partial Redeployment of Java EE Modules (In-Place Redeployment)	Module classloaders are immediately replaced with newer classloaders to load the updated class files. WebLogic Server does not guarantee uninterrupted client access to the module during redeployment, and existing clients' state information may be lost.	• Replacing a component of an enterprise application that has been taken off-line for scheduled maintenance, or that does not require uninterrupted client access.

Table 1 shows a summary of each strategy and good examples of where to use each strategy.

5.3.1 In-Place Redeployment

In-Place redeployment is a redeployment strategy where the whole application is redeployed immediately causing current user sessions to expire. The user will see that if an application requires login he or she will be logged out. And to continue working user needs to login again. Also, if there were any unsaved data that users have entered, those are lost. Obviously, this is not good behaviour from the user perspective. To avoid that users can be informed manually that there is a service break but nowadays users do not expect this kind of behaviour from the application. Depending on what kind of update it

is going in the application message to users can be more user friendly. For that purpose, Oracle WebLogic server offers two different In-Place redeployment strategies.

5.3.2 Partial Redeployment of Static Files (In-Place Redeployment)

The partial redeployment of static files is In-Place redeployment strategy where just static files are redeployed instead of the whole application. Static files are usually web application files such as static HTML pages, JSPs and graphics. Compared to the whole application In-Place redeployment this method does not affect existing user sessions. This means that user usually does not even know that the application was updated. Disadvantage of this method is that the application must be deployed in exploded format. It means that the file system files must be in directory structure instead of archived (i.e. zip) format.

5.3.3 Partial Redeployment of Java EE Modules (In-Place Redeployment)

The partial redeployment of Java EE modules is also an In-Place redeployment strategy where a single module or the subset of modules from the enterprise application is redeployed. This is a very similar approach to the partial redeployment of static files. The only difference is that the WebLogic server does not guarantee uninterrupted client access. That means the user will most likely see a similar, usually very small, interruption or outage while the whole application is redeployed. This method also assumes that the application is deployed in the exploded format in the file system.

5.3.4 Production Redeployment

The Production redeployment is a strategy to redeploy applications without causing interruption to the users. This is the Oracle WebLogic server way to provide application high availability. In this strategy, two versions of the same application can be running at the same time. Old version of that application serves existing client connections and new version serves all new client connections. When the last client connection to the old version terminates, old application will undeploy itself and there is only one version (new version) running. After that application must be removed manually from WebLogic. That is necessary because there can be only two versions of the same application installed in WebLogic server on the same time.

In Figure 5 is displayed the WebLogic server production redeployment when there are old and new versions running at the same time.

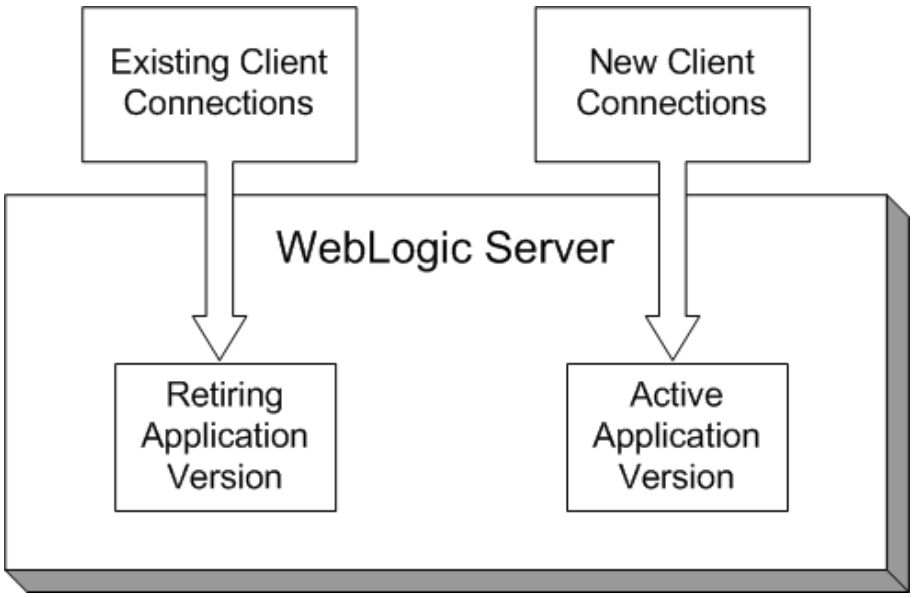

Figure 5. Production Redeployment (Oracle, 2015, pp. 8-4)

To achieve this feature the application must have a version number for the WebLogic server. Using that, WebLogic server can conclude that the new deployed application is a different version and decide it should make production redeployment. There are also restrictions when and where production redeployment can and cannot be used. Oracle documentation (Oracle, 2015, pp. 8-4,8-5) lists these requirements.

The production redeployment strategy is supported for:

- *Standalone Web Application (WAR) modules and enterprise applications (EARs) whose clients access the application via a Web application (HTTP).*
- *Enterprise applications that are accessed by inbound JMS messages from a global JMS destination, or from inbound JCA requests.*
- *All types of Web Services, including conversational and reliable Web Services, but not 8.x Web Services.*
- *Coherence cache clients – Grid archive (GAR) modules that are deployed within an EAR to storage-disabled managed Coherence servers. Classes in the GAR must be backward compatible with the existing deployment.*

Production redeployment is not supported for:

- *Standalone EJB or RAR modules. If you attempt to use production redeployment with such modules, WebLogic Server rejects the redeployment request. To redeploy such modules, remove their version identifiers and explicitly redeploy the modules.*
- *Applications that use JTS drivers. For more information on JDBC application module limitations, see JDBC Application Module Limitations in Administering JDBC Data Sources for Oracle WebLogic Server.*
- *Applications that obtain JDBC data sources via the DriverManager API; in order to use production redeployment, an application must instead use JNDI to look up data sources.*
- *Applications that include EJB 1.1 container-managed persistence (CMP) EJBs. To use production redeployment with applications that include CMP EJBs, use EJB 2.x CMP instead of EJB 1.1 CMP.*
- *Coherence cache servers – GAR modules that are deployed to storage-enabled managed Coherence servers.*

In production redeployment (and other deployment methods also) there is one extra measure what can be done before launching a new version of the application. And it is very beneficial to use that in conjunction with production redeployment. It is deploying the application in administration mode. This mode means that the application is deployed to the server, but external clients cannot access it. Only clients using administration channel can access it.

In Figure 6 application deployment via administration channel is shown.

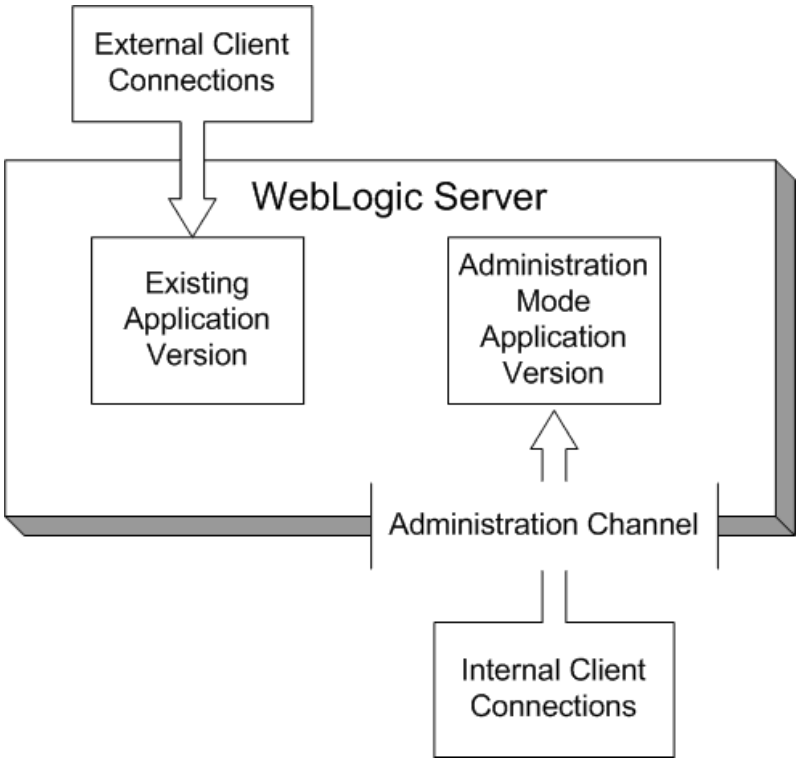

Figure 6. Distributing a New Version of an Application (Oracle, 2015, pp. 8-11)

In Figure 6 is shown the difference between administration channel and normal application usage. Administration channel provides a different route to the newer application version.

5.4 Current Deployment Process

To understand the deployment process, the current build pipeline needs to be investigated first. The build pipeline contains everything from the issue tracker to releasing software to the production. The issue tracker contains all necessary info to develop required coding task. A developer takes the issue and starts working on it. After finished working he/she tests it and commits changes to the version control system. The continuous integration (CI) server polls the version control system and if it finds new changes it starts the unit test task. CI runs the unit tests and if any failures are found it will notify the developer by email. If any failures are not found, build pipeline is ready for performing the integrations tests. These tests are run once a day. Every night the CI system polls new changes and starts tests if new changes are found. The integration tests are long running tests and that is why they are run only once a day.

Depending on the integration test results the next step is manual testing (if all tests have passed). Or back to developer (if there are test failures). Before the manual testing someone builds a release candidate and starts the deployment process to the testing environment. The deployment process is not an automated process so the user must go to the continuous integration server and select the desired build plan and start it. After the compilation is ready the user downloads the artefact to his/her own computer. Then the artefact is uploaded to the server where installation is going to be made. In this case it is the testing environment. After that, the user opens the application server administrator console, selects the uploaded artefact and starts the installation process. The installation process is WebLogic's internal process and it is not relevant to the present study and is therefore not described here. The deployment process is described in a document called Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server (Oracle, 2015).

After deployment to the test environment, testers start to perform the manual tests. The purpose of running these tests is to be certain that the fix or new feature is working as expected. But maybe the most important thing in manual testing is to test that the new feature (or fix) is easy and intuitive to use. Automating this kind of testing is impossible as for the time being. Manual testers also try to break the fix but usually they concentrate on its look and feel. If everything is working as expected the next phase in the pipeline is to start the deployment process to the production environment on selected time. The production deployment follows the same pattern than the testing environment deployment except there is no new build for the release candidate (or anything else). Instead the previously compiled application is renamed to the final version and that is used on the deployment. In Figure 7 is shown the current deployment pipeline and its every step.

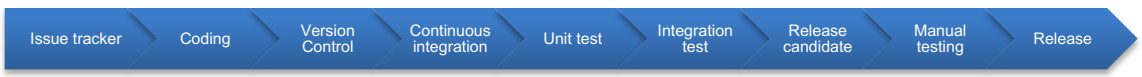

Figure 7. Current deployment pipeline

When analysing the current build pipeline, two main problems can be found. First, almost every step requires human interaction, building release candidate, downloading / uploading and deploying. The second problem is that the integration tests are long running and cannot be run several times a day. The third problem is that deploying the application causes a break to users. To overcome these three issues the deployment process needs to be automated by scripting, the integration tests and its environment should be altered to run tests faster and the latter issue can be solved using WebLogic production redeployment feature.

5.5 WebLogic Server Software Upgrade

Every software has flaws, even those which do not have updates. They just have not been found yet. That is why Oracle WebLogic server itself also needs updates from time to time. The WebLogic server 12.1.3 does not support continuous availability out of the box when talking about updating server itself. This is a new feature in the WebLogic server 12.2.1. There are also several other features for continuous availability and the most important ones are listed here (Oracle, 2016, pp. 2-7):

- **Automated cross-domain transaction recovery** - Provides automatic recovery of XA transactions across an entire domain, or across an entire site with servers running in a different domain or at a different site.
- **Zero Downtime Patching** - Provides an automated mechanism to orchestrate the rollout of patches while avoiding downtime or loss of sessions.
- **WebLogic Server Multitenant live resource group migration** - Provides the ability to migrate partition resource groups that are running from one cluster/server to another within a domain without impacting the application users.
- **Coherence federated caching** - Replicates cache data asynchronously across multiple geographically distributed clusters.
- **Coherence GoldenGate HotCache** - Detects and reflects database changes in cache in real time.
- **Oracle Traffic Director** - Routes HTTP, HTTPS, and TCP traffic to application servers and web servers on the network.
- **Oracle Site Guard** - Enables administrators to automate complete site switchover or failover.

The most interesting new feature in the previous list is Zero Downtime Patching (ZDT). It makes it possible to patch WebLogic Server across a domain while the application continues serving clients without interruption. Although WebLogic Server has supported rolling upgrades since version 9.2, the process has always been manual (Oracle, 2016, pp. 2-23). In this new version patching can be fully automated and there is a possibility to decide how many nodes needs to be patched. Patches are rolled out one node at a time and load balancer can redirect traffic to nodes that are not under patching. After patching is done load balancer will redirect traffic to newly patched node. (Oracle, 2016, pp. 2-23) This feature is really waited but unfortunately this is not available on WebLogic 12.1.3 which is in use. That is why there is a need for a different approach.

In most of the cases it is possible to patch one WebLogic server at time. This makes it possible to do patching without interrupting the clients. Oracle provides a tool called OPatch to do patching. OPatch is a Java-based utility that runs on all supported operating systems and requires installation of the Oracle Universal Installer. It is used to apply patches to Oracle software. (Oracle, 2014, p. 1)

OPatch is used to apply Patch Set Updates (PSUs) or single patches periodically. Oracle releases patches to its products four times in a year. After every released a patch (set) there is a decision process where decision is made should any patch to be installed or not. Usually individual patches are not installed but with some critical issues single patches are also installed. In the patch description, there is mentioned how the patch should be installed. As stated before usually a single patch or PSU does not need the whole cluster shutdown. Instead of that the upgrade can be made one managed server at a time. With this approach the user does not see interruptions.

This was a brief overall description on OPatch and what can be done with it. More detailed information is described in Oracle Fusion Middleware Patching with OPatch. (Oracle, 2014)

6 Implementation

The implementation was started with exploring the application and investigating if it was production redeployment ready. Source code was ploughed through for unsupported technologies and techniques. The codebase was relatively new because there was heavy refactoring a couple of years ago and it was easy to go it through. Fortunately, no incompatibilities were found so the production redeployment implementation could be started for the application.

6.1 Implementing Production Redeployment

To make the application production redeployment ready it is necessary to specify the application version identifier. Oracle recommends storing a unique version string directly in the MANIFEST.MF file of the EAR or WAR being deployed (Oracle, 2015, pp. 8-6). The recommendation was followed so one step was added to the build process for making that change. Building the application was done using Maven so the addition was quite simple. Maven Archiver plugin was configured to specify the version identifier automatically. Oracle gives strict rules on how the version identifier must be specified. The rules are described in Table 2 below.

Table 2: Valid and Invalid Characters. (Oracle, 2014, pp. 7-4)

Valid ASCII Characters	Invalid Version Construct
a-z	..
A-Z	.
0-9	
period (“.”), underscore (“_”), or hyphen (“-“) in combination with other characters	

It was decided that the application version identifier format was to follow the following format: application version number (including dot), underscore date and time with minutes and seconds. Below is example from MANIFEST.MF:

```
Weblogic-Application-Version: 1.11_20160612134531
```

The application version is configured elsewhere in Maven pom.xml. Maven plugin uses that version to construct the application version identifier. The date part is just formatted build time and it is taken from the machine where the application is built. With these

changes WebLogic server understands that the application can be deployed using production redeployment. Below is a snippet from pom.xml which defines the WebLogic application version.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ear-plugin</artifactId>
  <version>${version.maven.ear.plugin}</version>
  <configuration>
    <archive>
      <manifestEntries>
        <WebLogic-Application-
Version>${application.version}_${timestamp}</WebLogic-
Application-Version>
      </manifestEntries>
    </archive>
  </configuration>
</plugin>
```

The snippet describes Maven to use its EAR-plugin to create ear-file and modify its MANIFEST.MF file by adding entry “WebLogic-Application-Version”. The entry value is constructed from two parts. The first part is the “application.version”. This is a predefined application configuration variable and when ear-plugin is creating MANIFEST.MF file this “application.version” is substituted with the real value. The second part is the “timestamp”. It is substituted with the current date time value when Maven build is run. The “timestamp” value is an application property in which the value comes from “maven.build.timestamp” variable. There is a need to use different property because there is a bug when using with filtering. The timestamp format is also defined in the properties. It was defined to be “yyyyMMddHHmmss”.

6.2 Implementing Deployment Process Change

There were two main problems in the deployment process. The first one was that the deployment process needs human interactions in almost every step. The second problem was how to make developers to change their development process and especially how often they should commit changes to version control system. To

overcome the first issue, it was necessary to identify which step can be automated. As explained in the previous chapter everything starts after the user commits his or her changes to the version control system. The continuous integration server is automatically polling changes and if found the integration tests are started. After that human interaction is needed. Currently the test environment deployment is not started automatically because deployment is done using the Administrator console.

Fortunately, WebLogic server supports application deployment to be scripted so there is no real need to use Administration console to perform deployment. The WebLogic application server offers WLST (WebLogic Scripting Tool) which can be used to script the deployment process. More info about WLST can be found from WLST manual (Oracle, 2017).

The deployment phase scripting is started from automatically copying the release candidate from the continuous integration (CI) server to the testing environment. That is done by shell scripting. When the CI server has built a release candidate version it triggers the shell script which copies package to the test environment. When the release candidate is located on the testing server the actual deployment can be started. For that Python script was created. That script lists available packages to deploy and asks the user which one it should use for deployment. Then it asks the username and password which is used in deployment. The last step is to ask the target environment and after that it starts the production redeployment to that environment.

Another issue was developer develop process changes. Before process changes developers were committing their changes once a day or once in a several days, mainly when their task was finished and ready to review. To support the production redeployment there was a need to make developers commit more often, preferably several times a day. The production redeployment runs two different application versions at same time. If there is for example database changes, the changes must be compatible for both application versions. Sometime this is not possible, for example, when there is need to change a database column type. This can be done by making the change in small pieces. First a small commit adds a new column to the database. Then the second commit converts the existing the data to the new column. Also, the application is changed to use the new column. At the same time the database trigger must be made to copy any new data from the old column to the new column. Then the third commit will check is there any data in the old column which is not in the new column and finally removes the

old column. Every commit is installed separately to the production so there are three production installs in one feature. Along the automated tests production the install process becomes a very quick operation. That makes it possible to install several application versions to the production for example in a day. In Figure 8 is shown the refined deployment pipeline.

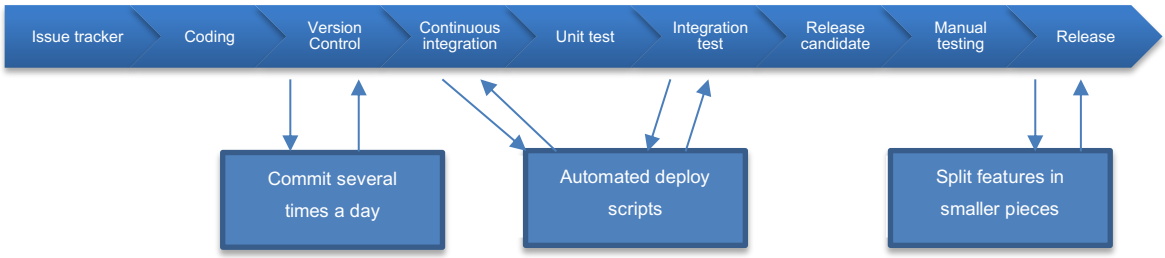

Figure 8. Modified deployment pipeline

The pipeline itself did not get any new steps. Modifications were done inside the pipeline steps. In the version control step developers changed their commit strategy to commit more often. Automated deploy scripts were added to continuous integration and integrations test steps and the actual production releases were significantly reduced in size.

6.3 Implementing Continuous Integration Changes

The continuous integration server was configured to poll the version control system periodically. When it notices a change, the server starts the configured build steps. In this case there were two steps: running the tests and starting and stopping the Docker container. Running the tests build step calls Maven with a new test profile. The profile was configured to run the test against supported browsers, Internet Explorer 8, Internet Explorer 9, Internet Explorer 10, Internet Explorer 11 and Chrome. Starting and stopping the Docker container was run after all the tests. It just ensures that out of memory or similar exceptions are not seen when deploying the application to WebLogic several times.

The biggest continuous integration problem was the unreliable long running tests. To improve that all the tests need to be checked and see what are the main reasons why

they are failing. Fortunately, the unreliable tests were restricted to user interface tests. The unit tests were reliable and there was not a need to change those.

On the other hand, the user interface tests were harder to fix. Usually there was no single test or single line which was failing the test. But a common factor was that test framework did not find the component from the page. The test framework in use is Selenium and it was a common problem that Selenium did not find the component from the page. There were several ways what could be done to fix the problems. Usually those were just hacks which could help in one place but not in another. One common fix was to add hard wait time to the tests. This was working quite well almost everywhere. A downside in this is that it makes the tests to run even longer than now. And the most of the time it was just useless waiting.

Eventually the failing tests were written differently. Checks of what was expected to be found in the page were changed and useless waits were removed. It was quite a tedious task to do but it was worth it. The tests become quite reliable after fixing. There are still some places where the test may occasionally fail. But in overall the tests work better now.

The next step was to increase the browser coverage. Before the tests were run only with Internet Explorer 8. That was the oldest browser supported. Because the tests were working better now it was easy to add Internet Explorer 9, Internet Explorer 10, Internet Explorer 11 and Chrome to the test runs. It was done by using the containers. The container technology in use was Docker. The setup was one cloud server for Docker and one cloud server per browser.

The continuous integration server was a key part in the environment. It builds the application and deploys it to the Docker server. In the Docker server, there are as many containers as there are browsers. After that it initiates the test run for each browser. Each browser will call the Docker server and there it dedicated instance of the application. In Figure 9 there is the overall picture of the automated testing environment.

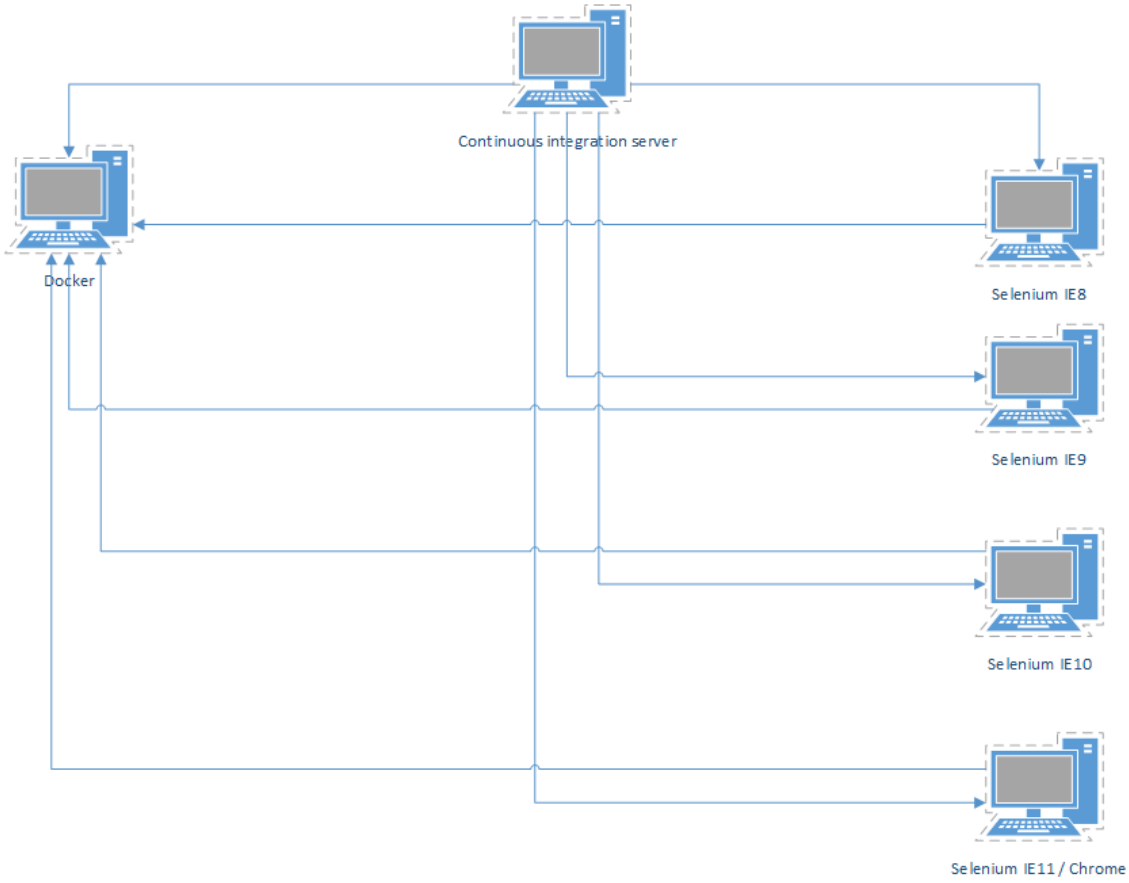

Figure 9. Automated testing environment

Oracle offers a template to build custom WebLogic Docker images. That was taken for starting point and it was customized heavily to match needs. For the time of building the solution Oracle did not offer Docker image for Oracle Database and that was the reason why the current in memory (H2) database was used instead of the same database which is running in the production. Currently Oracle offers also Database images for Docker and the chance to use that was added to the roadmap. The test did not cover every integration there are in the production environment but still there were quite many components that needed to include the Docker image. That made building the image quite hard a task to do.

The Docker image construction was started with Oracle's template. It was a really simple template which creates a simple domain and one server instance. The template uses Oracle WebLogic Docker image and that needed to be created first. Fortunately, Oracle offers good documentation for that because WebLogic binaries are not publicly available, and the binaries needed to be downloaded first to be able to run the build script. But

again, it was well documented so this was not a hard task to do. After creating the WebLogic base image, it was time to customize the Oracle's Docker template.

Customizing was started by installing necessary utility software to the Docker image, i.e. less, telnet, unzip, etc. The base images are usually made to be light weight as possible. That is why utility software needs to be to be installed separately. After that WebLogic needs to be patched to match the production WebLogic patch version. It was done by adding patch commands to the image creation. After that all necessary external libraries, required by software, were added to the image. Then was the time to modify the domain and server instance creation. They were modified to match the production domain. Of course, it was much simpler than in the production but there were still some configuration to do.

After these modifications, it was time to actually tweak the server start-up parameters. Memory settings were adjusted and the administrator user login name was added. To support multiple browser instances, there was a need to modify the application data sources accordingly. That was done by creating additional Docker images for each browser instance. It uses previously created Docker image and adds data sources to it.

The following sums up the construction of the Docker images:

- Create WebLogic base image from Oracle
- Customize Oracle's Docker templates
- Create one image for each browser instance using customized template

The script for starting the Docker containers was constructed after building the Docker images. The testing platform in use was Arquillian and it was used for deploying and running the tests. The benefit of using Arquillian was that there is no need to do anything else than configure the Docker container addresses, Arquillian will handle the rest.

The environment where the Docker is running is a virtual Linux server in the cloud. The Docker hosts multiple containers, one for each browser test. Each container listens to a different port so they are not interfering each other and they can run parallel.

6.4 Implementing Release Process Changes

The current release process was quite old fashioned, as described earlier. There was also need for a faster release cycle which is also describer earlier. First, releases were made smaller. It could be as small as one commit if necessary. With this and the WebLogic production redeployment releases can be done more often. In practise, the release cycle was changed to cover one or two individual changes and after that the new release was made.

This was quite a dramatic change compared to the old one. Earlier a new application version was prepared for manual testing for two weeks. Now the testing time was much shorter. The comprehensive tests were made previously to cover regressions and with those it was quite clear that any old features were not going to break. The only thing to do was to test a new feature / bug fix. Then the release was ready for the production installation. Installations were performed in the evenings, not by night, which was the previous practice thanks to the WebLogic production redeployment. The decision to make deployments to the production only in the evenings was just a precaution. The most active usage time was not wanted to be interfered if something should go wrong. The usage is much lighter in the evenings than it is at the day time.

For the production installation the following steps were performed:

- Commit changes to version control system
- Run automated tests in continuous integration server
- Install release build to testing environment
- Manual test release
- Manual install release to production

6.5 Design Guideline

The design guideline was rather easy to do after studying the production redeployment. Oracle has listed supported application configurations and from the environment side there was not any real requirements. So, the design guideline is just a shortened version of Oracle's list. The design guideline is listed in Appendix 1. The guideline summarizes the requirements (supported configurations) for the production redeployment and what

techniques should not be used. It is really straightforward to check those if the checker is the developer of the application and knows what technologies are used there.

The basic principal is that if standard and commonly used Java EE technologies are used everything should be fine. Legacy technologies such as EJB 1.1 CMP should be avoided.

7 Results and analysis

This chapter provides the test results for the redeployment tests. It also includes an explanation to how the deployment process was changed.

7.1 Testing without Production Redeployment

For testing, if the production redeployment is working, a JMeter script was developed to test session creation and application login. Basically, the script was trying to login to the application and perform search function there. The tests were performed on the testing environment where WebLogic Server was running and the application in it. The JMeter script was running on separate laptop. First, the application was updated without the production redeployment which means interruption to the user because the application was stopped, removed, installed and started. Just to show that the application was actually stopped the JMeter script was also running in this case. Table 3 below shows the results.

Table 3: Summary result without production redeployment

Samples	Average	Error %	Throughput	Received kB/sec	Avg. Bytes
40 000	36	84.76	161.6 / sec	862.9	5464.1

As can be seen in the summary table above, 40 000 sample requests were made. Average elapsed time (Average column) was 36 milliseconds and error percent (Error % column) was 84.76. Error percent is the most interesting column in this table. It shows how many percent of requests failed. This column varies the most between runs because it depends on how fast deployment can be run. Rest of the columns just shows how fast the handled requests was (Throughput column), how much kilobytes was received (Received kB/sec column) and average size of the sample response (Avg. Bytes column).

As Table 3 shows, there is a service break when updating the application without the production redeployment. This was a no-brainer to realize because the application was stopped, undeployed, deployed and started in this approach. That is also the reason why the test was run only once instead of five runs.

7.2 Testing with Production Redeployment

Next the production redeployment test was done. The application was altered to have the previously described application version identifier. The production redeployment process differs from the normal interrupted deployment process. There is no need to stop and start the application when it is updated. Update action is not available there if the application version identifier is not specified. So, the deployment process was to select the application, choose the update and run the update. This update process was performed five times and Table 4 shows the results.

Table 4: Summary result with production redeployment

#	Samples	Average	Error %	Throughput	Received kB/sec	Avg. Bytes
1	40 000	135	0.00	69.8 / sec	1448.48	21 249.2
2	40 000	138	0.00	69.1 / sec	1433.10	21 247.2
3	40 000	189	0.00	47.3 / sec	981.56	21 245.2
4	40 000	305	0.00	32.0 / sec	663. 73	21 244.3
5	40 000	143	0.00	66.6 / sec	1382.06	21 246.6

As can be seen in Tables 3 and 4, there is a huge gap between when using the production redeployment or not. When the production redeployment is not used there is always a break. But when the production redeployment is used there is not any break. Other column values vary depending network traffic and the server load.

In addition to specific tests for the production redeployment, every deployment to the development and the test environment were done using the production redeployment. Issues related to the deployment were not discovered. In all there were almost a couple of hundreds production redeployments before application were deployed to the production.

Oracle WebLogic server does what it promises. The web application which fills Oracle's requirements from the production redeployment can be updated without a service break.

7.3 Testing New Deployment Process

The deployment process changes were not that big in practice. The continuous integration server was configured to meet new process. The change was bigger to the mind-set, the trust to continuous integration and especially the trust to the fixed tests was hard to get. Before the tests were quite unreliable and the tests were often failing. Now there was almost zero fails and one just need to trust that everything was working.

A new build step was added to the continuous integration server to run the integration tests after every code commit. This way it is clear that every build is a release candidate which can be installed to the production. The downside here was that this increased the build time dramatically. The usual build time was between a couple of minutes to five minutes, depending on the load of the continuous integration server. Now the build time was between one to two hours. Luckily it was not a big problem because the application was in quite a matured state and that is why there was just a few changes in a month.

One build step was added to the build plan after the successful builds. This step creates a release version of the application and it is copied to the development / testing environment. That way it is easy to install whatever version is wanted. This takes quite a lot of disk space and for that a cleaning task was developed to remove old (over 2 months) packages.

8 Discussion and Conclusions

The goal of this thesis was to make an existing Java EE environment and one application to be continuously available. The goal was also to make the design guideline how the new application should be developed (Humble & Farley, 2011) or the existing application should be transformed so that it can be used in the WebLogic Server environment and with the production redeployment. The purpose of the guidelines was to help transform other applications to the production redeployment ready. The scope of the study was limited just to one application even if there are multiple applications running on that environment.

Oracle's production redeployment is a WebLogic server feature which makes it possible to update an application without interfering users. This means that users will not see any breaks when updating the application if the requirements are fulfilled. All users who are currently using the system use the old version of the application until their session ends. All the new users are redirected to the new version of the application. After all users end their sessions with the old version of the application, WebLogic server automatically shuts down the old application and only the latest version of the application is running.

Transforming the application to the production redeployment ready was quite trivial. The application already met Oracle's production redeployment requirements. The actual configuration change was also easy because Maven supports such changes out of the box. It was only necessary to add one line to the application MANIFEST.MF file.

Another thing to make the application continuous available was the deployment process change. The current deployment process was quite traditional where the application releases were quite big and they were installed to the production once a month. The process was changed to support instant product installation. Now the integration tests are run after every commit instead of running them only once a day. This increased the build time dramatically but it is manageable because there are not that many changes in the application.

For future development, there is a clear need to make the integration tests faster. That makes it even easier to release new versions of the application to the production. If there is a problem in the production, after fixing it, running the integration tests takes just too much time for fast fix. Developing the application will also become more meaningful when

there is no need to wait for the tests to be run. If the new development task is started and the tests fail there is a need to go back to the previous task. And that is not good for developer coding flow and motivation.

To make the environment support continuously available even more, one option could be to update WebLogic server to a newer version, equal or higher than 12.2.1. This version of WebLogic server supports updating the server itself to the new version. Currently patching can be done one server at a time only if the patch supports it. Fortunately, most of the patches do. With limited time and budget, WebLogic version upgrade was postponed to roadmap.

References

Alcott, T., 2010. *The WebSphere Contrarian: High availability (again) versus continuous availability.* [Online]
Available at:
http://www.ibm.com/developerworks/websphere/techjournal/1004_webcon/1004_webcon.html
[Accessed 17 August 2016].

Apache Software Foundation, 2015. *Apache JMeter.* [Online]
Available at: http://jmeter.apache.org
[Accessed 31 January 2016].

Gupta, A., 2012. *Java EE 6 Pocket Guide by Arun Gupta.* [Online]
Available at: https://www.safaribooksonline.com/library/view/java-ee-6/9781449338329/ch01.html
[Accessed 21 November 2017].

Humble, J. & Farley, D., 2011. *Continuous Delivery: reliable software releases through build, test, and deployment automation.* Boston: Pearson Education, Inc.
Oracle, 2005. *Scalability and High Availability.* [Online]
Available at: https://docs.oracle.com/cd/B14099_19/core.1012/b13994/avscalperf.htm
[Accessed 21 November 2017].

Oracle, 2014. *Fusion Middleware Patching with OPatch.* [Online]
Available at: https://docs.oracle.com/middleware/1213/core/OPATC/toc.htm
[Accessed 07 July 2017].

Oracle, 2014. *Oracle® Fusion Middleware High Availability Guide.* [Online]
Available at: https://docs.oracle.com/middleware/1213/core/ASHIA/toc.htm
[Accessed 25 January 2016].

Oracle, 2014. *Oracle WebLogic Server 12.1.3 is Release.* [Online]
Available at:

https://blogs.oracle.com/WebLogicServer/entry/oracle_weblogic_server_12_11
[Accessed 24 09 2016].

Oracle, 2015. *Java Platform, Enterprise Edition (Java EE).* [Online]
Available at: http://www.oracle.com/technetwork/java/javaee/overview/index.html
[Accessed 13 March 2016].

Oracle, 2015. *Oracle® Fusion Middleware Deploying Applications to Oracle WebLogic Server 12.1.3.* [Online]
Available at: https://docs.oracle.com/middleware/1213/wls/DEPGD/toc.htm
[Accessed 28 January 2016].

Oracle, 2016. *Oracle® Fusion Middleware What's New in Oracle WebLogic Server 12.2.1.* [Online]
Available at: https://docs.oracle.com/middleware/1221/wls/NOTES/toc.htm
[Accessed 07 July 2017].

Oracle, 2017. *Fusion Middleware Understanding the WebLogic Scripting Tool 12.1.3.* [Online]
Available at: https://docs.oracle.com/middleware/1213/wls/WLSTG/toc.htm
[Accessed 5 March 2017].

Schmidt, K., 2006. *High Availability and Disaster Recovery: Concepts, Design, Implementation.* 1st Edition ed. New York: Springer Berlin Heidelberg.
The Eclipse Foundation, 2017. *The Eclipse Enterprise for Java Project Top Level Project Charter.* [Online]
Available at: https://projects.eclipse.org/projects/ee4j/charter
[Accessed 29 Oct 2017].

Wikimedia Foundation, Inc., 2017. *Storage area network.* [Online]
Available at: https://en.wikipedia.org/wiki/Storage_area_network
[Accessed 04 July 2017].

Design guideline

This document describes what you need to check when you want to make your application production redeployment (for WebLogic application server) ready.

Production redeployment is supported for following configurations:

1. Application should be WAR, EAR and GAR (see bullet 5 for more info)
2. Application should be Web application, client access is done via HTTP
3. EAR with JMS, are accessed by inbound JMS messages from a global JMS destination, or from inbound JCA requests
4. All Web Services types except 8.x Web Services
5. Coherence cache clients - Grid archive (GAR) modules that are deployed within an EAR to storage-disabled managed Coherence servers. Classes in the GAR must be backward compatible with the existing deployment.

To ensure application will work, check that application does not use following features:

1. Standalone EJB or RAR module
2. JTS drivers
3. DriverManager API for obtaining JDBC data sources
4. EJB 1.1 CMP
5. Coherence cache servers - GAR modules that are deployed to storage-enabled managed Coherence servers

Detailed requirements and explanations can be found from: https://docs.oracle.com/middleware/1213/wls/DEPGD/redeploy.htm#DEPGD270

WebLogic application deploy script (sanitized)

```
from stat import ST_MTIME
import os
import time

def get_and_print_files_from_dir(directory, file_prefix):
    result = []
    file_entries = []
    for fn in os.listdir(directory):
        file_entries.append(os.path.join(directory, fn))

    stat_path_entries = []
    for path in file_entries:
        stat_path_entries.append((os.stat(path), path))

    entries = []
    for stat, path in stat_path_entries:
        # S_ISREG(stat[ST_MODE]) doesn't work with WLST because Java implementation
        # returns always 0
        if os.path.isfile(path) and os.path.basename(path).startswith(file_prefix):
            entries.append((stat[ST_MTIME], path))

    entries.sort(_compare_tuple)
    index = 0
    for cdate, path in entries:
        if os.path.basename(path).startswith(file_prefix):
            result.append(os.path.basename(path))
            print str(index) + ': ', time.ctime(cdate), os.path.basename(path)
            index += 1

    return result

def _compare_tuple(first, second):
    return cmp(int(first[0]), int(second[0]))

def deploy_app(admin_url, username, password, application_name, file_location,
target_cluster_name):
    print '*** WEBLOGIC : DEPLOY START ***'
    print 'Parameters: \nadmin_url: ' + admin_url + ', \nusername: ' + username + ',
\npassword length: ' \
          + str(len(password)) + ', \napplication_name: ' + application_name + ',
\nfile_location: ' \
          + file_location + ', \ntarget_cluster_name: ' + target_cluster_name
    print 'connecting to admin server....'
    connect(username, password, url=admin_url, adminServerName='AdminServer')
    print 'stopping and undeploying ....'
    stopApplication(application_name)
    undeploy(application_name)
```

```
    print 'deploying....'
    progress = deploy(appName=application_name, path=file_location,
targets=target_cluster_name, stageMode='stage',
                      remote='true', upload='true')
    progress.printStatus()
    print ''

    while progress.isRunning():
        progress.printStatus()
        print ''
        time.sleep(5)

    startApplication(application_name)
    print 'disconnecting from admin server....'
    disconnect()
    print '*** WEBLOGIC : DEPLOY STOP ***'

def prod_redeploy_app(admin_url, username, password, application_name, file_location,
target_cluster_name):
    print '*** WEBLOGIC : PRODUCTION REDEPLOY START ***'
    print 'Parameters: \nadmin_url: ' + admin_url + ', \nusername: ' + username + ',
\npassword length: ' \
          + str(len(password)) + ', \napplication_name: ' + application_name + ',
\nfile_location: ' \
          + file_location + ', \ntarget_cluster_name: ' + target_cluster_name
    print 'connecting to admin server....'
    connect(username, password, url=admin_url, adminServerName='AdminServer')
    print 'production redeploying....'
    progress = deploy(appName=application_name, path=file_location,
targets=target_cluster_name, stageMode='stage',
                      remote='true', upload='true')
    progress.printStatus()
    print ''

    while progress.isRunning():
        progress.printStatus()
        print ''
        time.sleep(5)

    print 'disconnecting from admin server....'
    disconnect()
    print '*** WEBLOGIC : PRODUCTION REDEPLOY STOP ***'


# =========MAIN PROGRAM ======================== #
mode = raw_input('What you want to do?\n1. WLS deploy\n2. WLS Production Redeployment\
\nEnter option number:')

environment = raw_input('Please enter WLS environment (prod/test/dev):')
```

```
if environment == 'prod':
    adminURL = 't3://127.0.0.1:7001'
    upload_directory = '/opt/oracle/admin/upload'
    print 'Production is not supported yet!'
    exit()
elif environment == 'test':
    adminURL = 't3://127.0.0.1:7101'
    upload_directory = '/opt/oracletest/admin/upload'
elif environment == 'dev':
    adminURL = 't3://127.0.0.1:7201'
    upload_directory = '/opt/oracledev/admin/upload'
else:
    print 'Invalid environment: ' + environment
    exit()

application = raw_input('Select application to update:\n\
1. Application\
\nEnter option number:')

if application == '1':
    application_name = 'app1'
    if environment == 'prod':
        target_cluster_name = 'app1Cluster'
    elif environment == 'test':
        target_cluster_name = 'app1testCluster'
    elif environment == 'dev':
        target_cluster_name = 'app1devCluster'
else:
    print 'No application selected, exiting...'
    exit()

print 'Select file to update:'

files = get_and_print_files_from_dir(upload_directory, application_name)

file_index = raw_input('Enter file number:')
file_name = files[int(file_index)]

confirm = raw_input('You selected file: ' + file_name + '. Is that correct (y/n)?')

if confirm != 'y' and confirm != 'Y':
    exit()

username = raw_input('Please enter your WLS username:')
password = raw_input('Please enter your WLS password:')

if mode == '1':
    deploy_app(adminURL, username, password, application_name, upload_directory +
os.path.sep + file_name, target_cluster_name)
elif mode == '2':
```

```
    prod_redeploy_app(adminURL, username, password, application_name, upload_directory +
os.path.sep + file_name, target_cluster_name)

print '...done'

exit()
```

Base Dockerfile (sanitized)

```
#
# This is modified version from Oracle sample Dockerfile (1213-domain) for Application
# https://github.com/oracle/docker-images/tree/master/OracleWebLogic
#
# ORACLE DOCKERFILES PROJECT
# --------------------------
# This Dockerfile extends the Oracle WebLogic image by creating an Application domain.
#
# Util scripts are copied into the image enabling users to plug NodeManager
# magically into the AdminServer running on another container as a Machine.
#
# REQUIREMENTS
# ------------
# You need to build oracle/jdk:8 and oracle/weblogic:12.1.3-generic first and add
# them to docker images. More info:
# https://github.com/oracle/docker-images/tree/master/OracleWebLogic
# But basically:
#      $ cp server-jre-8u111-linux-x64.tar.gz OracleJava/java-8
#      $ cp OraleJava/java-8
#      $ sh build.sh
#      $ cp fmw_12.1.3.0.0_wls.jar OracleWebLogic/dockerfiles/12.1.3
#      $ sh ./buildDockerImage.sh -v 12.1.3 -g
#
# Should look like this before building and running this image:
# [root@centos7-1 ~]# docker images
# REPOSITORY         TAG               IMAGE ID         CREATED             VIRTUAL SIZE
# oracle/weblogic    12.1.3-generic    94ab090fbbb0     48 minutes ago      2.201 GB
# oracle/serverjre   8                 b6fd289585e7     56 minutes ago      433.3 MB
# oraclelinux        latest            df602a268e64     3 weeks ago         276.2 MB
#
#
# HOW TO BUILD THIS IMAGE
# -----------------------
# Put all downloaded files in the same directory as this Dockerfile
# Put appserver-ear-5.7.3-weblogic.ear in the same directory as this Dockerfile
# Put appserver-ear-6.2.2-weblogic.ear in the same directory as this Dockerfile
# Put appserver-ws-1.3.2.ear in the same directory as this Dockerfile
# Put appserver-ws-2.0.ear in the same directory as this Dockerfile
# Put appserver-integration-4.1.1.ear in the same directory as this Dockerfile
# Put jrebel-7.0.12-nosetup.zip in the same direcotry as this Dockerfile
# Run:
#      $ sudo docker build -t app11213-domain --build-arg ADMIN_PASSWORD=welcome1 .
#
# HOW TO RUN
# ----------
# -p <local_port>:<container_port>
```

```
#      $ sudo docker run -d --name wlsadmin --hostname wlsadmin -p 7301:7301 app11213-
domain
#
# CONNECT CONTAINER
# -----------------
#      $ sudo docker exec -ti wlsadmin /bin/bash
#
# RUN WITH JREBEL
# ---------------
# JRebel is activated by default on ManagedServers. Do not try to use it with
AdminServer. It doesn't work.
# ManagedServer remote debugging ports can be found from add-server.py file.
# You also need JRebel license file (jrebel.lic). Put it to same directory where
Dockerfile is.
#
# RUN WITH REMOTE DEBUG
# ---------------------
# Run container with remote debugger
#      $ sudo docker run -d --name wlsadmin --hostname wlsadmin -p 7301:7301 -p
59701:59701 \
#        --env JAVA_OPTIONS=" -Dfile.encoding=UTF-8 -
agentlib:jdwp=transport=dt_socket,address=59701,suspend=n,server=y $JAVA_OPTIONS"
app11213-domain
#
# CREATE MANAGED (App1, APP2, appserver etc.) SERVERS
# ---------------------------------------
# Admin server (see above) must be running
#      $ sudo docker run -d --name app1-env --link wlsadmin:wlsadmin \
#        -p 7303:7303 -p 7321:7321 -p 7331:7331 -p 7307:7307 -p 7319:7319 -p 59721:59721
-p 59703:59703 -p 59731:59731 \
#        -e APP1_DS_PASSWORD=<database_password> \
#        -e APP2CMS_DS_PASSWORD=<database_password> \
#        -e APP2COMMON_DS_PASSWORD=<database_password> \
#        -e APP2RESTRICT_DS_PASSWORD=<database_password> \
#        -e APP3_DS_PASSWORD=<database_password> \
#        -e APPSERVER_DS_PASSWORD=<database_password> \
#        -e A_APPSERVER_DS_PASSWORD=<database_password> \
#        -e APP2REF1_DS_PASSWORD=<database_password> \
#        -e APP2REF2_DS_PASSWORD=<database_password> \
#        -e APP2REF3_DS_PASSWORD=<database_password> \
#        -e APP2REF4_DS_PASSWORD=<database_password> \
#        -e APP2REF5_DS_PASSWORD=<database_password> \
#        -e APP2REF6_DS_PASSWORD=<database_password> \
#        -e ADMIN_HOST=wlsadmin \
#        -e ADMIN_PORT=7301 \
#         app11213-domain createServer.sh

# If you want to mount local directory to container add following run parameter to run
command:
#      -v /c/Users:/u01/mount_dir
```

```
#

# Pull base image
# ---------------
FROM oracle/weblogic:12.1.3-generic

# Maintainer
# ----------
MAINTAINER Heikki Nykyri <heikki.nykyri@appelsiini.fi>

# WLS Configuration
# -----------------------------
ARG ADMIN_PASSWORD
ARG APP1_DS_PASSWORD
ARG APP2CMS_DS_PASSWORD
ARG APP2RESTRICT_DS_PASSWORD
ARG APP3_DS_PASSWORD
ARG APPSERVER_DS_PASSWORD
ARG A_APPSERVER_DS_PASSWORD
ENV DOMAIN_NAME="app1" \
    DOMAIN_HOME="/u01/oracle/user_projects/domains/app1" \
    ADMIN_PORT="7301" \
    ADMIN_HOST="wlsadmin" \
    ADMIN_USERNAME="weblogic" \
    NM_PORT="5556" \
    APP1_CLUSTER_NAME="App1DockerCluster" \
    APP2_CLUSTER_NAME="APP2DockerCluster" \
    APP3_CLUSTER_NAME="App3DockerCluster" \
    APPSERVER_CLUSTER_NAME="AppserverDockerCluster" \
    A_APPSERVER_CLUSTER_NAME="AApperverDockerCluster" \
    CONFIG_JVM_ARGS="-Dweblogic.security.SSL.ignoreHostnameVerification=true" \
    JAVA_OPTIONS="-Xmx1024m -XX:MaxPermSize=768m -Dfile.encoding=UTF-8" \
    PATCH_ZIP=p25869659_121300_Generic.zip \
    PATCH_NUMBER=25869659 \
    H2_JAR=h2-1.3.175.jar \
    APPSERVER_EAR=appserver-ear-5.7.3-weblogic.ear \
    APPSERVER_INTEGRATION_EAR=appserver-integration-4.1.1.ear \
    APPSERVER_WS_EAR=appserver-ws-1.3.2.ear \
    A_APPSERVER_EAR=appserver-ear-6.2.2-weblogic.ear \
    APPSERVER_WS_2_EAR=appserver-ws-2.0.ear \
    JREBEL_ZIP=jrebel-7.0.12-nosetup.zip \

PATH=$PATH:/u01/oracle/oracle_common/common/bin:/u01/oracle/wlserver/common/bin:/u01/ora
cle/user_projects/domains/app1/bin:/u01/oracle

# Istall tools for debugging etc. as root
USER root
RUN yum -y install less && yum -y install telnet && yum -y install wget && yum -y
install unzip
```

```
# Add files required to build this image
USER oracle
COPY container-scripts/* /u01/oracle/
COPY container-scripts/app/* /u01/oracle/app/
COPY container-scripts/app/appserver/* /u01/oracle/app/appserver/
COPY container-scripts/app/appserver/xml/* /u01/oracle/app/appserver/xml/
COPY container-scripts/app/appserver/xml/stylesheets/*
/u01/oracle/app/appserver/xml/stylesheets/
COPY container-scripts/app/aappserver/* /u01/oracle/app/aappserver/
COPY container-scripts/app/aappserver/xml/* /u01/oracle/app/aappserver/xml/
COPY container-scripts/app/aappserver/xml/stylesheets/*
/u01/oracle/app/aappserver/xml/stylesheets/
COPY container-scripts/app/app1/* /u01/oracle/app/app1/
COPY container-scripts/app/app3/* /u01/oracle/app/app3/
COPY container-scripts/app/app2/* /u01/oracle/app/app2/
COPY container-scripts/app/app2/templates/* /u01/oracle/app/app2/templates/
COPY container-scripts/app/app2/wsdl/* /u01/oracle/app/app2/wsdl/
COPY $PATCH_ZIP /u01/
COPY $H2_JAR /u01/
COPY $APPSERVER_EAR /u01/oracle/upload/
COPY $APPSERVER_INTEGRATION_EAR /u01/oracle/upload/
COPY $APPSERVER_WS_EAR /u01/oracle/upload/
COPY $AAPPSERVER_EAR /u01/oracle/upload/
COPY $APPSERVER_WS_2_EAR /u01/oracle/upload/
COPY $JREBEL_ZIP /u01/
COPY jrebel.lic /u01/

# Configuration of WLS Domain
WORKDIR /u01/oracle
RUN /u01/oracle/wlst -loadProperties /u01/oracle/datasource.properties
/u01/oracle/create-wls-domain.py && \
    mkdir -p /u01/oracle/user_projects/domains/app1/servers/AdminServer/security && \
    echo "username=$ADMIN_USERNAME" >
/u01/oracle/user_projects/domains/app1/servers/AdminServer/security/boot.properties && \
    echo "password=$ADMIN_PASSWORD" >>
/u01/oracle/user_projects/domains/app1/servers/AdminServer/security/boot.properties && \
    echo ". /u01/oracle/user_projects/domains/app1/bin/setDomainEnv.sh" >>
/u01/oracle/.bashrc && \
    echo "export
PATH=$PATH:/u01/oracle/wlserver/common/bin:/u01/oracle/user_projects/domains/app1/bin"
>> /u01/oracle/.bashrc && \
    cp /u01/oracle/commEnv.sh /u01/oracle/wlserver/common/bin/commEnv.sh && \
    mkdir -p /u01/mount_dir && \
    mkdir -p /u01/download && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app1Properties && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app1Logs && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app3Properties && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app3Logs && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Logs/temp && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Logs/incoming && \
```

```
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Logs/incoming && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Logs/outgoing && \
    mkdir -p /u01/oracle/user_projects/domains/app1/rMessages && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Properties/templates && \
    mkdir -p /u01/oracle/user_projects/domains/app1/app2Properties/wsdl && \
    mkdir -p /u01/oracle/user_projects/domains/app1/appserverProperties/xml/stylesheets
&& \
    mkdir -p /u01/oracle/user_projects/domains/app1/appserverLogs && \
    mkdir -p /u01/oracle/user_projects/domains/app1/aappserverProperties/xml/stylesheets
&& \
    mkdir -p /u01/oracle/user_projects/domains/app1/aappserverLogs && \
    cp /u01/oracle/app/appserver/*.properties
/u01/oracle/user_projects/domains/app1/appserverProperties && \
    cp /u01/oracle/app/appserver/xml/*.xsd
/u01/oracle/user_projects/domains/app1/app1serverProperties/xml && \
    cp /u01/oracle/app/appserver/xml/stylesheets/*.xsl
/u01/oracle/user_projects/domains/app1/appserverProperties/xml/stylesheets && \
    cp /u01/oracle/app/aappserver/*.properties
/u01/oracle/user_projects/domains/app1/aappserverProperties && \
    cp /u01/oracle/app/aappserver/xml/*.xsd
/u01/oracle/user_projects/domains/app1/aappserverProperties/xml && \
    cp /u01/oracle/app/aappserver/xml/stylesheets/*.xsl
/u01/oracle/user_projects/domains/app1/aaserverProperties/xml/stylesheets && \
    cp /u01/oracle/app/app1/*.properties
/u01/oracle/user_projects/domains/app1/app1Properties && \
    cp /u01/oracle/app/app3/*.properties
/u01/oracle/user_projects/domains/app1/app3Properties && \
    cp /u01/oracle/app/app2/*.properties
/u01/oracle/user_projects/domains/app1/app2Properties && \
    cp /u01/oracle/app/app2/*.xml /u01/oracle/user_projects/domains/app1/app2Properties
&& \
    cp /u01/oracle/app/app2/templates/*
/u01/oracle/user_projects/domains/app1/app2Properties/templates && \
    cp /u01/oracle/app/app2/wsdl/*
/u01/oracle/user_projects/domains/app1/app2Properties/wsdl && \
    cd /u01 && unzip $PATCH_ZIP && \
    cd /u01/$PATCH_NUMBER && /u01/oracle/OPatch/opatch apply -silent && \
    rm -rf /u01/$PATCH_NUMBER && \
    mv /u01/$H2_JAR $DOMAIN_HOME/lib/$H2_JAR && \
    unzip /u01/$JREBEL_ZIP -d /u01/ && \
    /u01/jrebel/bin/activate.sh /u01/jrebel.lic

# Expose Node Manager default port, and also default http/https ports for admin console
EXPOSE $NM_PORT $ADMIN_PORT $MS_PORT

WORKDIR $DOMAIN_HOME

# Define default command to start bash.
CMD ["startWebLogic.sh"]
```